2019

UGV DIRECTION CONTROL BY HUMAN ARM GESTURE RECOGNITION VIA DETERMINISTIC LEARNING

Xiaotian Chen
University of Rhode Island, xiaotian_chen@my.uri.edu

UGV DIRECTION CONTROL BY HUMAN ARM GESTURE RECOGNITION VIA DETERMINISTIC LEARNING

BY

XIAOTIAN CHEN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2019

MASTER OF SCIENCE THESIS

OF

XIAOTIAN CHEN

APPROVED:

Thesis Committee:

Major Professor Paolo Stegagno

Chengzhi Yuan

Richard J.Vaccaro

James Miller

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2019

ABSTRACT

With the recent development of machine learning and neural networks, different applications have been developed to improve the intelligence of unmanned robotic systems. In this thesis, we present a novel method for controlling an unmanned ground vehicle (UGV) by using a new machine learning technique, called deterministic learning [1], to learn and recognize four specifically designed body languages, which represent four corresponding moving directions (i.e., left, right, up, and down) of the controlled UGV. The Microsoft Kinect sensor is employed to collect the human body skeleton data, including (x, y, z) coordinates of the human arm joints, from which four specifically-designed features are extracted for neural network training. The discrete-time deterministic learning algorithm is then utilized to train the radial basis function neural networks (RBFNNs). The dynamics of the human arm waving motion is guaranteed to be accurately identified, represented, and stored as a RBF NN model with converged constant NN weights, which facilitates the rapid recognition purpose in the testing phase. In the testing phase, a set of estimators are built based on the database established in the learning phase, so as to conduct real-time rapid recognition of new in-coming gesture commands. The smallest error principle is used to decode the human intention, the decoded result will then be sent to the UGV through TCP/IP to control its moving directions. A full-integrated graphical user interface (GUI) has been developed based on Python programming language to demonstrate effectiveness of the proposed approach and illustrate the proposed experimental results.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisers Prof. Paolo Stegagno and Chengzhi Yuan. I am really grateful for their encouragement and inspiration during my graduate study. Also, I feel grateful for all the members from my ICRL lab, as for they gave a lot helps during this research. Finally, I want to thanks to my parents for supporting me to study abroad.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Motivation

Nowadays, robotic systems are becoming more and more powerful and intelligent, they could be used to accomplish more complex, less structured tasks and activities. However, some complex tasks may require human's simple instructions to guide the robot. As such, robotic systems need to interact with humans to understand the instructions, which are known as Human-Robot Interaction (HRI) [2, 3]. Since different humans could give the same information in different ways or styles, the robot needs to have a kind of learning capability to adapt to such variant information. One of the most commonly used methods for the human to deliver their intentions is by the hand gestures. In the recent researches, human body language gains a lot of attention. One type of human hand gestures is static gestures, in which human is standing still with a pose in front of a camera. The idea of decoding the human intention through hand gesture recognition is to recognize the finger's positions of one hand in a picture to understand the meaning, like "OK" [4, 5]. Another type of human hand gestures is dynamic gestures, like using a finger to write letters [6, 7] or walking gait [8, 9, 10]. This type of human body languages could be complex and difficult for a robot to understand. Furthermore, this kind of gesture information may take time to do recognition and the accuracy rate may not be high enough for real-time robot controlling. On the controlled robot side, it needs to have the ability to decode the human intention, which is the human's dynamic gestures in this thesis. In most of the recent researches, there are two approaches. One is to mount several motion detection sensors on the body [11]. This approach is expensive and the user needs to spend time and efforts on wearing the whole set of sensors. Another way is to use the camera to record the

motion video (a set of pictures), which can be commonly achieved by two type of cameras: Kinect sensor [12, 13, 14, 1] and leap motion sensor [15]. Kinect is capable of detecting the whole human body while Leap motion can only detect close gestures, which motivates us to adopt the Kinect sensor in this research.

1.2 Contribution of The Research

In this thesis, I will present a new real-time human-robotic controlling method by human arm gesture recognition using Kinect via deterministic learning. Since human prefer to use arm gestures to deliver information to others in long distance, like waving hand to say goodbye or using hand to point directions. Under that distance, the voice message may contain too much noise compared to using the arm gestures. Motivated by this, we specifically designed four gestures to represent the four directions of an unmanned ground vehicle's (UGV) movement, i.e., moving left, moving right, moving up (forwards), and moving down (backwards). The

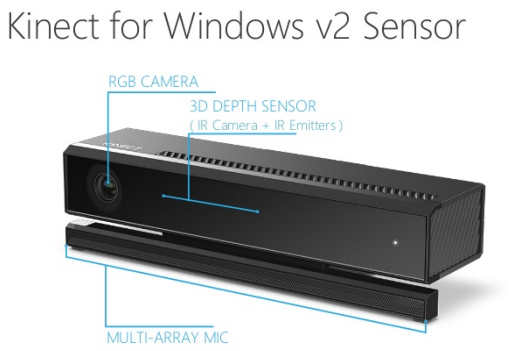

Figure 1: Picture of Kinect V2

Kinect sensor adopted in this research (as shown in figure 1) uses depth camera to capture the 3-D information of the environment and record the body gestures as depth image. Based on those pictures, the Microsoft Kinect is able to detect the coordinates (x, y, z) of 25 main skeletal joints of the human body. When the body is doing a repetitive arm gestures, those joints' motions can be view as period or

recurrent signals/trajectories. Thus, the discrete-time deterministic learning algorithm adopted from [16] can be implemented to approximate the dynamics of the human arm motion. In the training stage, the arm motion dynamics associated with a set of pre-given directional gestures will be approximated by radial basis function neural networks (RBFNNs). The obtained knowledge of approximated dynamics can be further stored in constant RBF networks with the NN weights guaranteed to converge to their ideal values. In the recognition stage, the unknown gesture will be compared to each dynamics in the obtained knowledge database. This gesture then will be recognized based on the well-known smallest error principle [17], in turn human intention decoding could be achieved in real time.

Another contribution of this thesis is that a fully-integrated graphical user interface (GUI) is built based on the Python programming language to make the operation process more convenient and user-friendly. The GUI is embedded the ability to connect the Kinect sensor and record the data, to run the neural network training and recognition process based on the discrete-time deterministic learning algorithms.

By using the gesture recognition results, we are able to control the moving direction of a real UGV. A small UGV is built in our lab with microprocessor and motors, which is able to move freely in a 2-D plane by giving velocity commands. A sample maze is created as the UGV running environment for better performing the result.

The whole procedure of this research is show in figure 2. It separates into five different phases: Phase one is the Kinect recording. Phase two is the feature extraction. Phase three is the deterministic learning training. Phase four is the deterministic learning recognition. Phase five is for commanding the robot. All

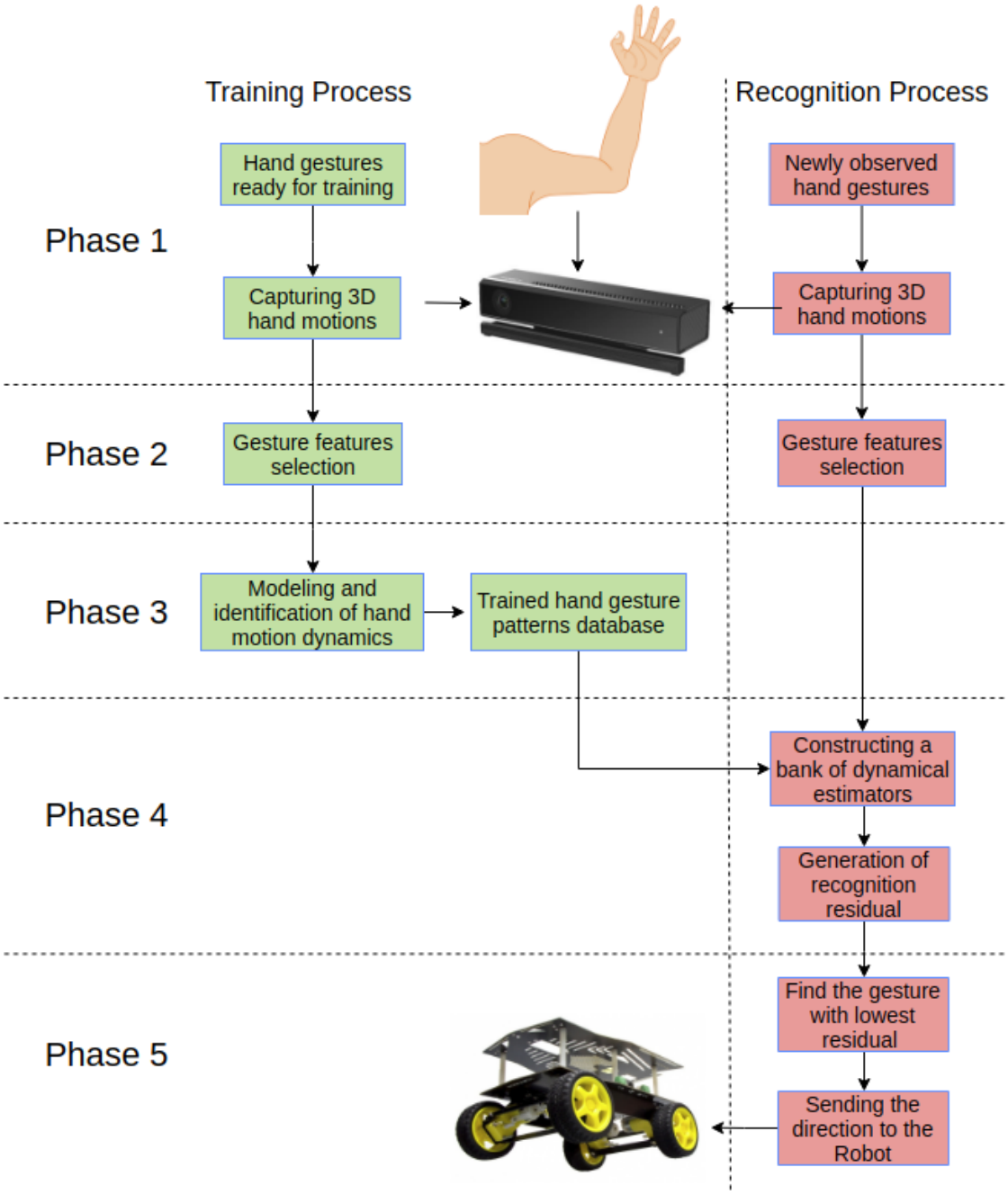

Figure 2: Block diagram of the proposed method for human-UGV interaction through arm gesture recognition with Kinect. Phase one is the Kinect recording. Phase two is the feature extraction. Phase three is the deterministic learning training. Phase four is the deterministic learning recognition. Phase five is for commanding the robot.

the phases will be introduced in Chapter 3 in detail. The Chapter 2 will give the preliminaries about the deterministic learning. Chapter 4 describers the setup and result of the experiments. Finally, Chapter 5 is the conclusion.

CHAPTER 2

Preliminaries About Deterministic Learning Theory

In deterministic learning theory [18, 19], identification of system dynamics of genneral nonlinear systems is achieved according to the following elements: (i) employment of localized RBF neural network; (ii) satisfaction of a partial persistence of excitation (PE) condition; (iii) exponential stability of the adaptive system along the periodic or recurrent orbit; (iv) locally accurate neural network (NN) approximation of the unknown system dynamcis .

The RBF neural networks can be described by $f_{nn}(Z) = \sum_{i1=1}^{N} w_i s_i(Z) = W^T S(Z)$, where $\mathbf{Z} \in \Omega_{\mathbf{Z}} \subseteq \mathbb{R}^q$ is the input vector, $W = \left[w_1, ..., w_N\right]^T \in \mathbb{R}^N$ is the weight vector. N is the NN node number, and $S(Z) = [s_1(||\mathbf{Z} - \mu_i||), ..., s_N(||\mathbf{Z} - \mu_i||)]^T$. With $s_i(\cdot)$ being a radial basis function, and $\mu_i (i = 1, ..., N)$ being distinct points in state space. The Gaussian function $s_i(||\mathbf{Z} - \mu_i||) = \exp\left[-\frac{(\mathbf{Z}-\mu_i)^T(\mathbf{Z}-\mu_i)}{\eta_i^2}\right]$ is one of the most commonly used radial basis functions, where $\mu_i = [\mu_{i1}, \mu_{i2}, ..., \mu_{iN}]^T$ is the center of the receptive field and η_i is the width of the receptive field. The Gaussian function belongs to the class of localized radial basis function s in the sense that $s_i(||\mathbf{Z} - \mu_i||) \to 0$ as $||Z|| \to \infty$.

It has been shown in [20] that for any continuous function $\int(Z) : \Omega_Z \to R$ where $\Omega_Z \subset R^p$ is a compact set, and for the NN approximator, where the node number N is sufficiently large, there exists an ideal constant weight vector W^*, such that for each $\epsilon^* > 0$, $f(Z) = W^{*T} S(Z) + \epsilon(Z)$, $\forall Z \in \Omega_Z$, where $|\epsilon(Z) < \epsilon^*|$ is the approximation error. Moreover, for any bounded trajectory $Z_\zeta(t)$ within the compact set Ω_Z, $f(Z)$ can be approximated by using a limited number of neurons located in a local region along the trajectory: $f(Z) = W_\zeta^{*T} S_\zeta(Z) + \epsilon_\zeta$, where $S_\zeta(Z) = [S_{j1}(Z), ..., S_{j\zeta}(Z)]^T \in R^{N_\zeta}$, with $N_\zeta < N, |s_{j_i}| > \iota (j_i = j_1, ..., j_\zeta), \iota > 0$ is

a small positive constant, $W_\zeta^* = [w_{j1}^*, ..., w_{j\zeta}^*]^T$, and ϵ_ζ is the approximation error, with $\Big||\epsilon_\zeta| - |\epsilon|\Big|$ being small.

Based on previous results on the PE property of RBF nrural networks [21] , it is shown in [22] that for a localized RBF network $W^T S(Z)$ whose centers are placed on a regular lattice, almost any recurrent trajectory $Z(t)$ can lead to the satisfaction of the PE condition of the regressor sub-vector $S_\zeta(Z)$. The class of recurrent trajectories includes periodic, quasi-periodic, almost-periodic, and even some chaotic trajectories, which comprise the most important types (though not all types) of trajectories generated from nonlinear dynamical systems (see [23] for a rigorous definition of recurrent trajectory)

CHAPTER 3

Method

In this chapter, we present a new scheme for human gesture recognition based control of UGVs. The core algorithm of this new scheme is inherited from the discrete-time deterministic learning theory [16]. The graphical user interface is built to facilitate the entire operation process.

3.1 Data Acquisition and Feature Extraction

The body gestures we present here is to wave the forearm pointing in different directions. As shown in Figure 3, for the forward gesture, the forearm will have the hand pointing direction and middle line along with the positive y-axis. The left gesture uses the negative x-axis. Right gesture uses the positive x-axis, and the backward gesture uses the negative y-axis. The hand will wave roughly between -45° and 45° of the middle line. The Kinect records the 3-D coordinate (x, y, z) of all the 25 joints in the rate of 30 frames per second, z is the distance to the Kinect camera. Only 6 data sets (three joints shown as three red points in Figure 3) are being used in our body gestures recognition: x and y coordinates of the central point of left hand (x_1, y_1), left elbow (x_2, y_2), and left shoulder (x_3, y_3).

The four features chosen to be used for the neural network training should be highly representing the human intention or the movement of the left arm, which include: (1) x-axis distance between left hand x_1 and left shoulder x_3; (2) y-axis distance between left hand y_1 and left shoulder y_3; (3) the angle between forearm and x-axis along the x-y plant; and (4) the angular velocity of the forearm. These four features can be mathematically calculated using the obtained $x-y$ coordinate data, i.e.,

$$d_x(k) = x_3(k) - x_1(k) \tag{1}$$

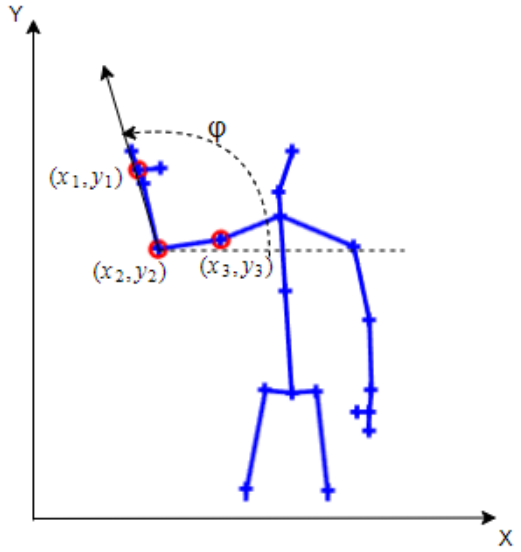

Figure 3: Human body skeleton extracted by Kinect

$$d_y(k) = y_3(k) - y_1(k) \tag{2}$$

$$\varphi(k) = tan^{-1}(\frac{y_2(k) - y_1(k)}{x_2(k) - x_1(k)}) \tag{3}$$

$$\omega(k+1) = \varphi(k+1) - \varphi(k) \tag{4}$$

For the third features, by using the arctan, the angle will be mapped to $[-\pi/2 : \pi/2]$. However one problem here is that one of the direction (in this experiment is right) will have the angle jump between $-\pi/2$ and $\pi/2$. The angular velocity will be extremely large at that point, which is undesirable. To solve this problem, we create a angular velocity value detection. If its corresponding absolute value is larger than a certain point (a threshold is set to be 3 in this project), all the angles in this pattern which are smaller than zero will be increased by 2π to recalculate the angular velocity. Examples for the resulted features are shown in Figures 4 and 5.

3.2 Discrete-time Deterministic Learning

In this section, the overall ideas of the Deterministic learning algorithm will be presented. The Deterministic learning mainly has two parts: learning and

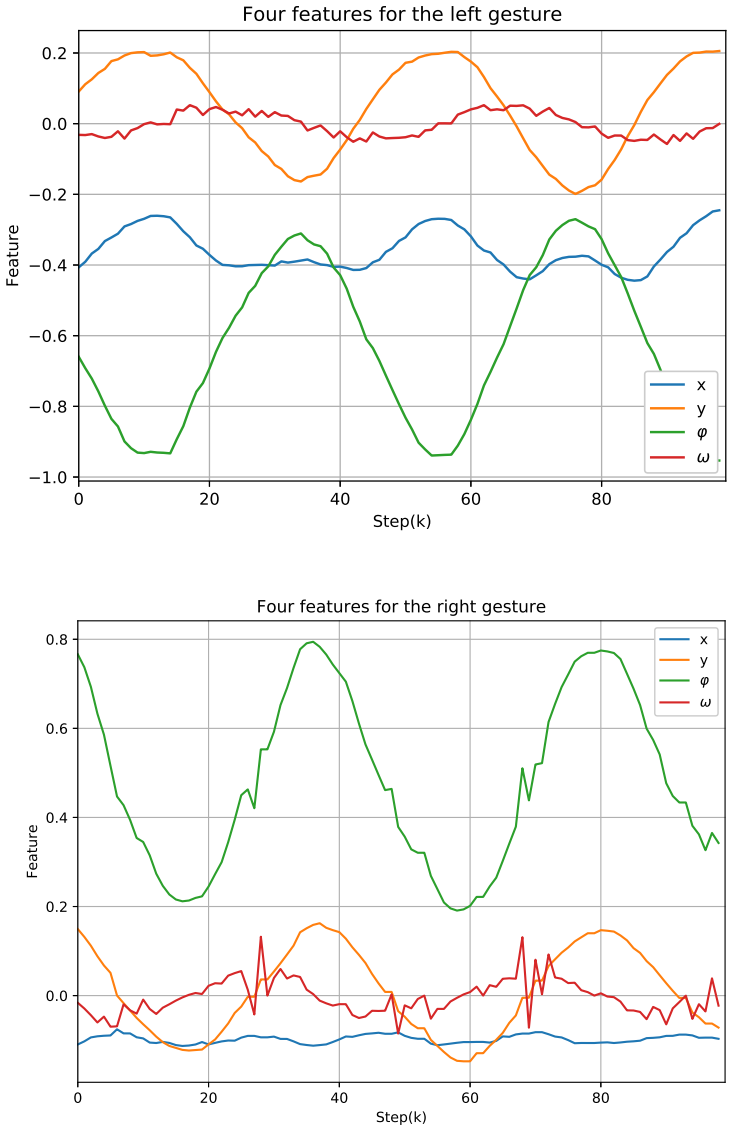

Figure 4: The sample features of $d_x(k), d_y(k), \varphi(k), \omega(k)$ for the gestures representing left and right (normalized between -1 to 1)

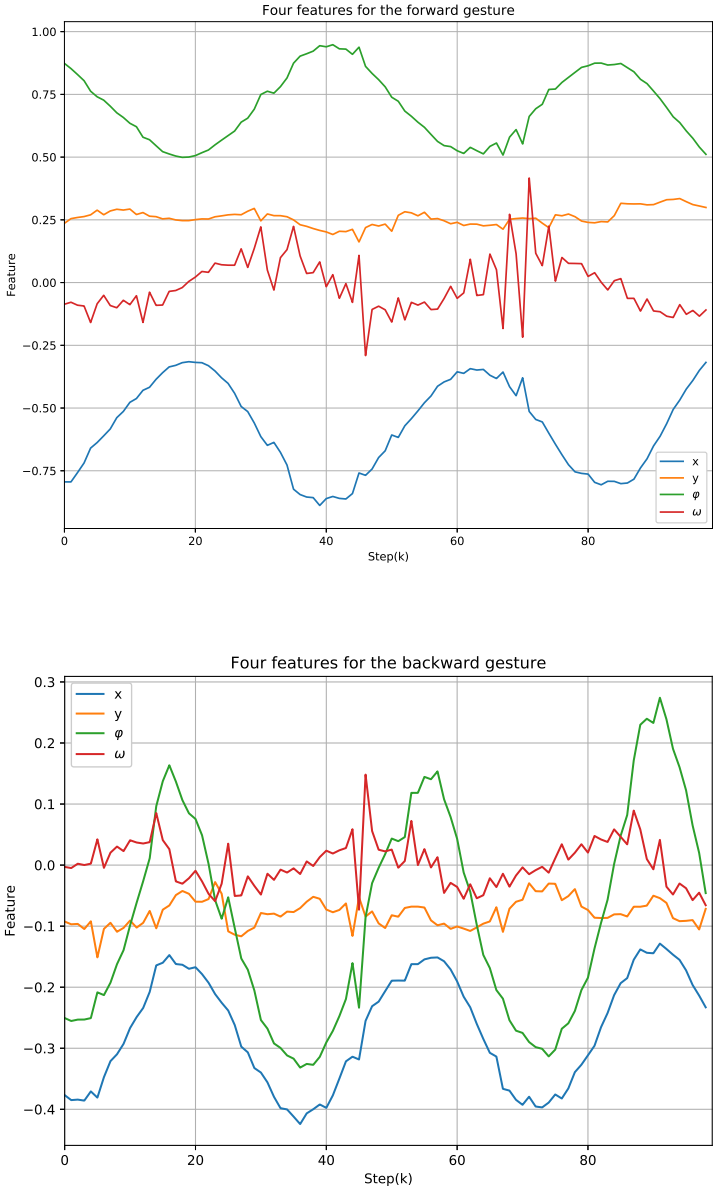

Figure 5: The sample features of $d_x(k), d_y(k), \varphi(k), \omega(k)$ for the gestures representing forward and backward (normalized between -1 to 1)

recognition.

3.2.1 Accurate Learning from Discrete-time Gesture Patterns

First of all, in order to let the robot to understand the gestures, we need to train the robot by the gestures with known directions. We implement the discrete-time deterministic learning algorithm to model the gesture data. The general nonlinear motion dynamics can be described in the form of

$$\dot{x} = F(x; p) \tag{5}$$

where $x = [x_1, ..., x_n]^T \in R^n$ is the state of the system with initial condition $x(t_0) = x_0$, which is representing the features of the gesture, n is the dimension of the features. p is the parameter of the system (constant vector). $F(x; p) = [f_1(x; p), ..., f_n(x; p)]$ gives a smooth but unknown nonlinear vector field which represents the overall motion dynamcis system. Since the data recorded by Kinect camera is a sampled data set with a sampling rate around $f_s = 30Hz$, it would be more realistic to use the discrete-time functions to model the system.

According to the ideas in [16], we build up the discrete-time deterministic learning model to test the body gestures. By using the Euler approximation, a discretized system representation of (5) can be written as

$$x[k+1] = x[k] + T_s f(X[k]; p), \qquad x[0] = x_0. \tag{6}$$

Based on the discrete-time deterministic learning theory [16], the dynamical RBF neural network can be employed to identify the motion dynamics of human gesture system represented in (6):

$$\hat{x}[k+1] = x[k] + a(\hat{x}[k] - x[k]) + T_s \hat{W}^T[k+1] S(X[k]), \tag{7}$$

$\hat{x} = [\hat{x}_1, \hat{x}_2, ..., \hat{x}_n]$ is the state vector of the dynamical RBF neural networks. Where $0 < |a| < 1$ is a design constant. And the $\hat{W}^T S(X[k])$ is used to approximate the

unknown function $f(X[k];p)$. Then in order to let the neural network to converge to the unknow function, The neural network weight's updating law to adjust the NN weights $\hat{W}$ is given by

$$\hat{W}[k+1] = \hat{W}[k] - \frac{\alpha P(e[k] - ae[k-1])S(X[k-1])}{1 + \lambda_{max}(P)S^T(X[k-1])S(X[k-1])}, \tag{8}$$

where $P = P^T > 0$, and $\lambda_{max}(P)$ denotes the largest eigenvalue of the matrix P, and α selected within the range from 0 to 2 is the learning gain to be designed. And $e[k] = \hat{x}[k] - x[k]$.

From the equations (7) and (6), we can infer that $e[k+1]$ satisfies the following equation:

$$e[k+1] = ae[k] + T_s\tilde{W}^T[k+1]S(X[k]) - T_s\epsilon \tag{9}$$

where $\epsilon = f(X;p) - W^{*T}S(X)$ is the NN approximation error. The RBF network weight error equation can be written as

$$\begin{aligned}\tilde{W}[k+1] &= \hat{W}[k+1] - W^* \\ &= \left[I - \frac{\alpha T_s P(S^T X[k-1]S(X[k-1]))}{1 + \lambda_{max}(P)S^T(X[k-1])S(X[k-1])}\right]\tilde{W}[k] \\ &\quad + \frac{\alpha T_s P S(X[k-1])\epsilon}{1 + \lambda_{max}(P)S^T(X[k-1])S(X[k-1])}.\end{aligned} \tag{10}$$

From equation (9) and (10), we can derive the perturbed LTV form as:

$$\begin{aligned}\begin{bmatrix} e[k+1] \\ \tilde{W}[k+1] \end{bmatrix} &= \begin{bmatrix} a & T_s S^T(X[k])A[k] \\ 0 & A[k] \end{bmatrix}\begin{bmatrix} e[k] \\ \tilde{W}[k] \end{bmatrix} \\ &\quad + \begin{bmatrix} T_s\epsilon(\dfrac{A[k]X[k]}{X[k-1]} - 1) \\ B[k] \end{bmatrix} \\ A[k] &= I - \frac{\alpha T_s P(S^T X[k-1]S(X[k-1]))}{1 + \lambda_{max}(P)S^T(X[k-1])S(X[k-1])} \\ B[k] &= \frac{\alpha T_s P S(X[k-1])\epsilon}{1 + \lambda_{max}(P)S^T(X[k-1])S(X[k-1])}\end{aligned} \tag{11}$$

From the Theorem 1 in [16], if the sampling period T_s satisfies $0 < T_s < 2/\alpha$, we can get that the state estimation error (equation (9)) will converge to a small

neighborhood of zero and also the $\hat{W}_\zeta$ will converge to its optimal value W_ζ^*. From these two results, we can obtain that the RBF network $\hat{W}^T S(\varphi_\zeta)$ can be used to locally-accurate approximate the unknown internal dynamic $f(X;p)$, i.e.,

$$f(X[k];p) = \bar{W}^T S(x) + \epsilon_2. \tag{12}$$

$$\bar{W} = \frac{1}{k_b - k_a + 1} \sum_{k=k_a}^{k_b} \hat{W}[k]. \tag{13}$$

where $k_a, ..., k_b$ represents a time segment after the transient process.

3.2.2 Rapid Recognition of Discrete-Time Gesture Patterns

In this subsection, we present a scheme for rapid recognition of dynamical sequences of the arm gestures. Following the results of [17], consider the training set containing the gestures patterns φ_ζ^s, $s = 1, ..., M$, with the sth training pattern φ_ζ^s generated from:

$$X^s[k+1] = X^s[k] + T_s F^s(X^s[k];p^s), \quad X^s[0] = X_{\zeta 0}^s. \tag{14}$$

where $X^s[k] = [x_1^s[k], ..., x_n^s[k]]^T \in R^n$ is the system state. $F^s(X^s[k];p^s)$ describe the motion dynamics of the arm gesture system which is an unknown nonlinear vector field with p^k being the system parameter vector. Similar to the training phase, this unknown system can be accurately identified and stored in the RBF networks $(\bar{W}^s)^T S(X[k])$. By constructing the estimators for each training pattern as:

$$\begin{aligned} \bar{X}^s[k+1] = \bar{X}^s[k] + T_s B(\bar{X}^s[k] - X[k]) \\ + T_s(\bar{W}^s)^T S(X[k]). \end{aligned} \tag{15}$$

where the $\bar{X}^s[k] = [\bar{x}_1^s[k], ..., \bar{x}_n^s[k]] \in R^n$ is the state of the estimators for all M patterns. $X[k]$ is the state of an input set pattern which is the one going to be recognized. $B = diag\{b_1, ..., b_n\}$ is a diagonal matrix which holds constant value

for all estimators, and it satisfies $-\frac{1}{T_s} < b_i < 0$. By inserting the testing pattern $f_i'(X[k];p')$ to each estimator, we obtain the recognition error system:

$$\begin{aligned}\tilde{x}_i^s[k+1] = (1+b_iT_s)\tilde{x}_i^s[k] + (\bar{W}_i^s)^T S_i(X[k]) \\ -f_i'(X[k];p').\end{aligned} \tag{16}$$

where the $\tilde{x}_i^s[k] \triangleq \frac{\bar{X}_i^s[k] - x_i[k]}{T_s}$ is the synchronization error. In order to correctly obtain the error, we use an average L_1 norm for the synchronization error to make the decision:

$$\|\tilde{x}_i^s[k]\|_1 = \frac{1}{T_e}\sum_{j=k-T_e}^{k-1} |\tilde{x}_i^s(j)|\,, \quad k > T_e \tag{17}$$

where T_e is a constant value representing the number of cycle of one certain gesture.

The whole idea for the recognition of the arm gestures is that we compare all $\|\tilde{x}_i^s[k]\|_1$, $s \in \{1,...,M\}$ to each other. Find out the sth estimator which has lowest norm error $\|\tilde{x}_i^s[k]\|_1$ among others. Then the testing pattern is said to have the same direction as the sth estimator.

3.3 The GUI Design

With the GUI developed in this project (see Figure 6 to 9), the user can easily control the entire machine learning process, including data collection, NN training, and real-time recognition as well as UGV control. The holistic discrete-time deterministic learning algorithm is integrated into this GUI. This GUI is build based on the Python programming language with Tkinter package. The whole GUI is mainly separated into four parts: collect, filter, learning, and recognition.

The first page is the collect page (Figure 6). This page is combined with three parts. The top side of the page is for the user to start Kinect and choose what kind of information to be displayed in the plot. The first button is to connect the Kinect, the user can start the recording. During the recording, the user can switch the plot between features or x-y-z positions of any joint. the graph will

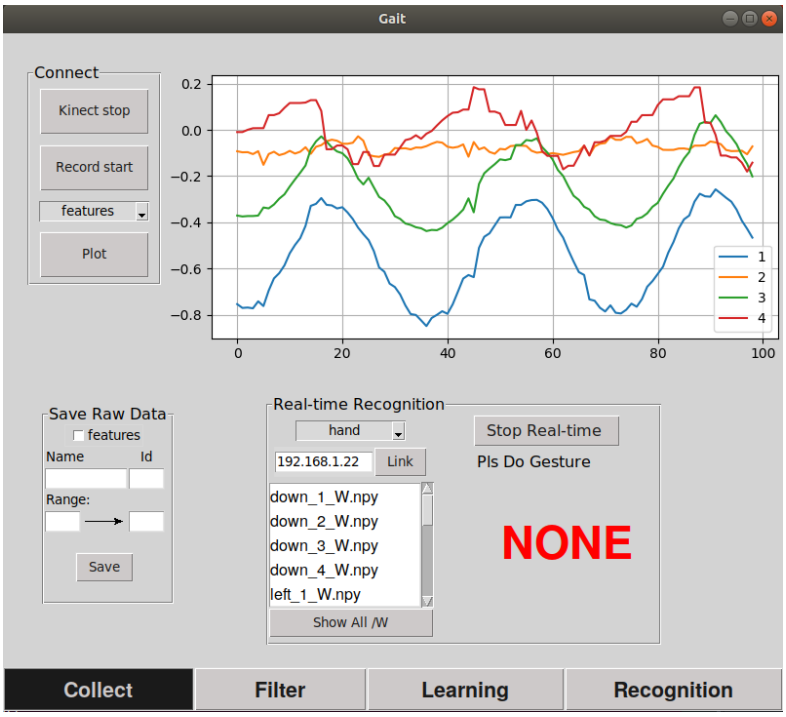

Figure 6: The GUI page for Kinect recording and real-time recognition

Figure 7: The GUI page for feature extraction

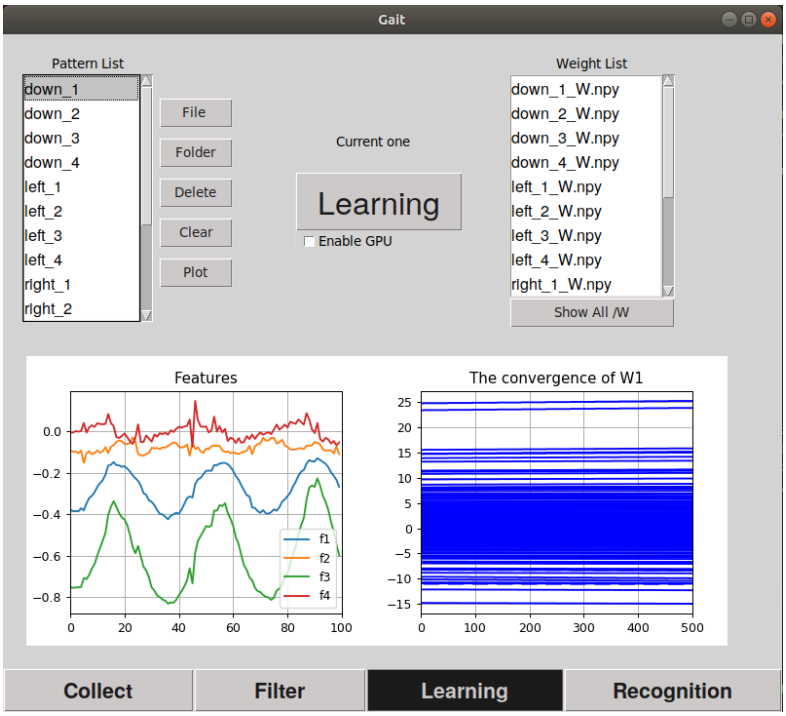

Figure 8: The GUI page for applying deterministic learning

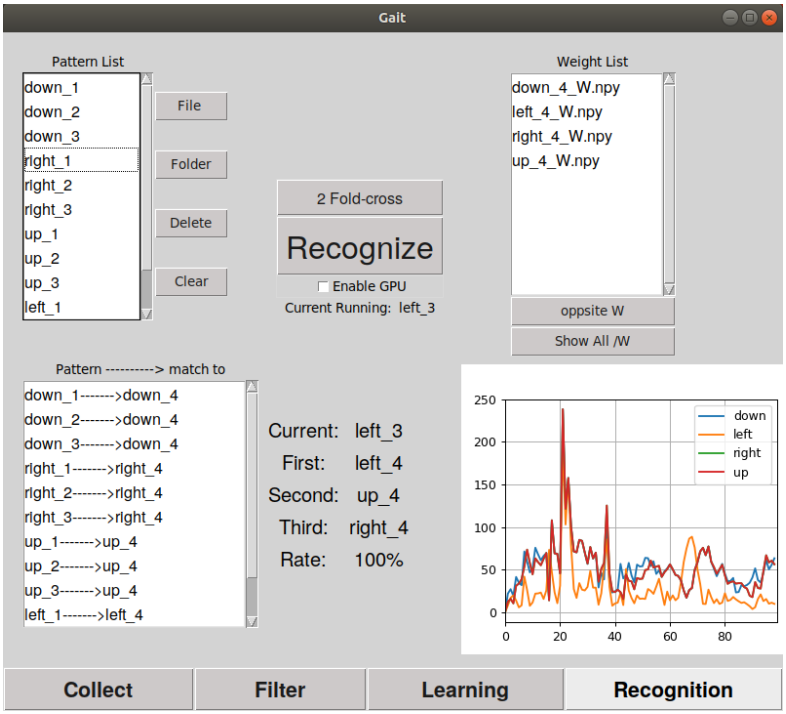

Figure 9: The GUI page for recognition testing

display the information for the last 3 seconds and keep running along the time. After the recording is stopped, by clicking on the “plot” button, the selected data from the last record will be displayed on the screen. Then, the user can move to the bottom left side of the page to save the data. By specifying the group name (the direction of the gesture just recorded), mark number, and range of the data (since there could be some garbage data at beginning or ending) , the data will be saved into the “jointdata” folder. Once the user select desired features, the joint data will be automatically converted to features and save as a file in “feature” folder. The rest of the page is for real-time recognition and UGV commanding. After choosing the learned patterns, filter type, and neural network configuration, the user can press the real-time recognition after Kinect is successfully connected. Then, the software will use the current 90 frames (about 3 seconds) of the recorded data, which is the part displayed on the axis to be recognized. After the real-time recognition finished, the result will be displayed in red, and also be sent as a direction signal to the linked IP address which should be the desired UGV.

The second page is for filtering the raw data and converting the filtered results to four features for deterministic learning (Figure 7). Currently, only hand gesture filter is implemented here, which converts the joint position data direct record from the camera to the four features. The user can use the ”TRY” button to display the result for the selected pattern in the list to the second plot. In which the bottom left is for displaying the unfiltered pattern. The ”RUN” button will convert all the patterns in the list and save as files under the folder named: features.

The third page is implemented with the learning process of discrete-time deterministic learning (Figure 8). By including any desired pattern to the learning list, the program will train the RBF NNs through deterministic learning. The GPU mode can be activated to accelerate the learning speed. After the learning

phase is finished, the software will show the converge figure, which allows the user to check if the learning is successful or not.

The last page is for recognition (Figure 9). In this page, the user is able to test the learned pattern and get the success rate. The learned weight files will be inserted into the top-left list, and the newly in-coming feature patterns will be put into the top-right list. Then, the user can start recognition (choose with GPU acceleration or not). The program will find the three most-matched learned patterns for every testing pattern through the recognition algorithm, and will record all the first match to the result list for every testing pattern. Then, the software will compare the name tag of the testing pattern with the most similar learned pattern, if they hold the same name (the same direction gesture), the recognition of this testing pattern will be marked as successful. Users can also see the computed errors for the latest recognition result.

CHAPTER 4

Experimental Studies

In order to test the performance of the discrete-time deterministic learning for the human gesture recognition, two different types of experiments will be performed in this chapter: pattern cross-validation recognition and commanding a simulator/real UGV.

4.1 Experimental Setup

The Kinect V2 can record two types of image: color image and depth image. Here in this research, only depth camera is being considered. The depth camera has a resolution of 512 $\times$ 424 pixels with a FOV (Field of View) of 70.6 $\times$ 60 degrees. For the body tracking, the Kinect suggests to have the maximum range of 4.5 meters. As shown in Figure 10, in this research, for the recording and real-time recognition, the Kinect camera is set 2.4 meters apart from the gesture demonstrator and 1.2 meters above the ground. However, since all the four features are normalized to the body position, the camera position won't have too much effect on the gesture features. So from the specification above, the tester's position theoretically can be at 2.1 meters to 4.5 meters away from the camera and 1.7

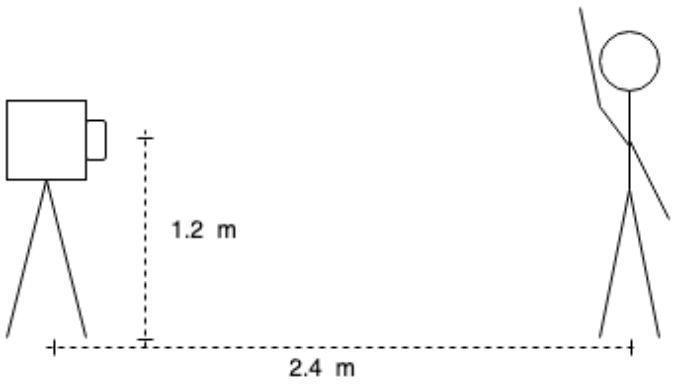

Figure 10: Schematic diagram of the Kinect position setup

meters to left or right of the middle line at 2.4 meters distance.

Overall, 40 human arm gesture patterns were recorded by Kinect for each direction, leading to 160 patterns in total. All patterns are demonstrated by the same person, but in a different date, which the second half of the patterns were recorded about one month later than the first half which may have some position or hand waving speed difference. Each pattern contains about 3 seconds of information with 90 data points (Kinect takes depth pictures at 30 fps). Every pattern is used to train the neural network and generate one weight file.

For the deterministic learning algorithm in this experimental, the RBF Neural Network has total number of the node $N = 83521$, which the centers μ_i are evenly spaced on $[-1.2 : 0.15 : 1.2] \times [-1.2 : 0.15 : 1.2] \times [-1.2 : 0.15 : 1.2] \times [-1.2 : 0.15 : 1.2]$. The design constant in equations (7) and (8): a is set to 0.5, α is set to 1.5, $P = diag\{10, 10, 10, 10\}$ which leads to $\lambda_{max}(P) = 10$. All the features which are used for deterministic learning are repeated for 300 times along the timeline, in this case, the weight will get enough time to converge to constant values. Figure 11 shows one of the examples of training result (in this figure, only shows 100 node weights with largest slopes during the last 1000 steps). All the RBF NN weights were learned for 27000 steps (every pattern is 90 steps and is repeated 300 times). The final weights we use is from the end of the converging. Taking the average of the last 1000 steps for each node and treat it as the final weights for the dynamic system. As we can see, there is some of the nodes not perfectly converging to the constant values, which may be caused by the noise during the recording and it also leads to the mismatch of the real features (x) and approximated features value ($\hat{x}$) as shown in Figure 13.

For the recognition part, the parameter b_i in equation (16) is set to be -15, and T_e in equation (17) is a value depending on the pattern length which makes

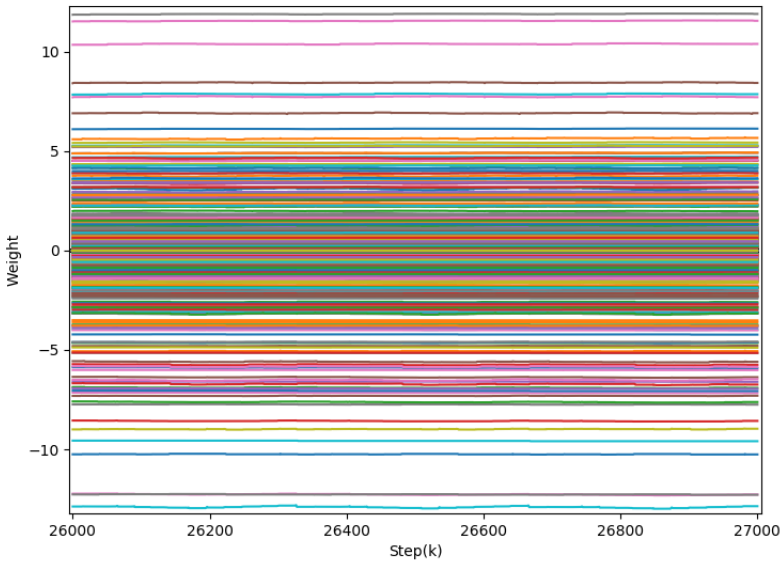

Figure 11: The convergence of $\hat{W}_3$ in the last 1000 steps during the learning process on the Right gesture pattern of the third feature. Only 100 weights with largest slopes are plot here.

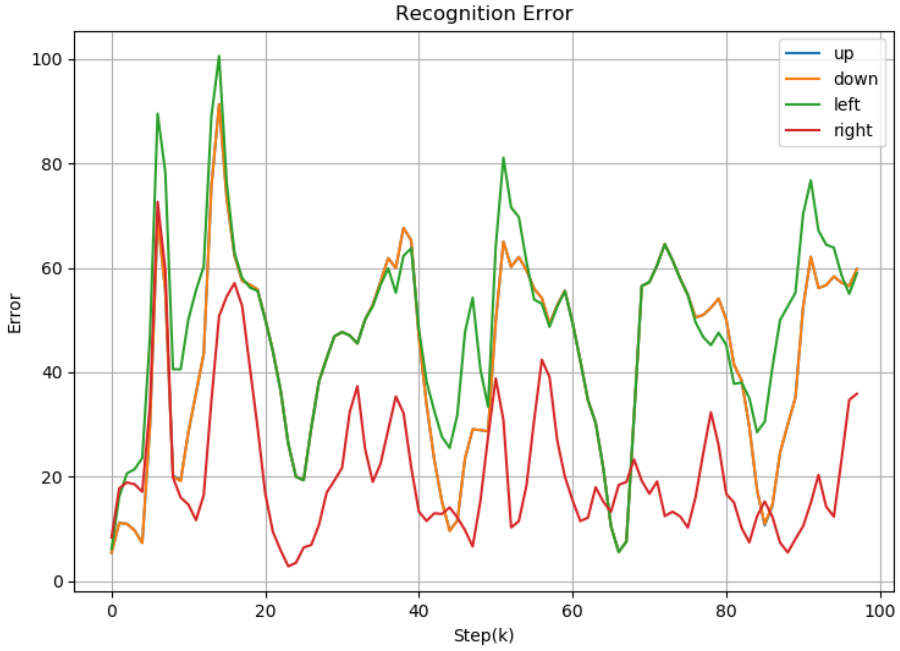

Figure 12: The example of the recognition error by testing the right gesture with the whole dynamic data-base

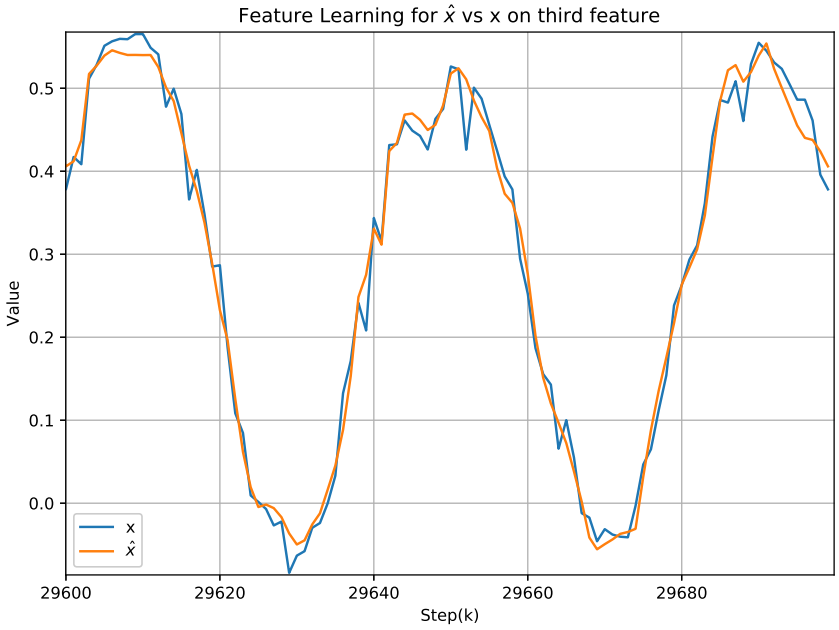

Figure 13: The comparing between the accurate state values and approximated state values

$(k - T_e)$ equal to 10. Figure 12 gives an example of the computed result of the synchronization error $\tilde{x}^s[k]$ in equation (16).

In order to decrease the recognition time during the real-time recognition for controlling the robot, we picked 8 training patterns (two for each direction) that have the highest recognition rate during the cross-validation test.

4.2 Recognition Rate Testing

To test the recognition result, we randomly separated all 160 patterns into 10 folders (each direction has 4 patterns in every folder). For the 2-folder cross-validation, the first five folders of training pattern were inserted into the learned data list in the recognition page of the GUI, and the other five folders of untrained patterns were put into the second list. Then switch the two list to test the successful rate again. For the 10-folder cross-validation, 9 folders will be used to recognize the other one folder. All 12 recognition results have 100% successful rate. The

Recognition type	Gallery Size	Probe Size	Correct Rate(%)
2-fold cross	80	80	100
10-fold cross	144	16	100
Real-time Recognition pattern	8	152	100

Table 1: Recognition results on four directional gestures

success rate for the 8 special patterns we picked up for fast real-time recognition is also 100%. By using those 8 training patterns to differentiate the rest 152 patterns. The table 1 shows the recognition rate of these three different types of recognition.

4.3 Unmanned Ground Vehicle Control

In this experiment, two types of ground vehicles are controlled by the real-time gesture command. One is the Gazebo simulated UGV and the other is a real UGV.

4.3.1 Real Ground Vehicle Plant

The ground vehicle plant we are using is a four-wheel drive vehicle, a simple schematic diagram is shown in Figure 14. The real UGV looks like Figure 15. All four wheels of the vehicle are fixed in the same direction. Once the computer on the vehicle receives the commanded direction signal, it will run in that direction for three seconds and stop to wait for the next command.

The four-wheel drive is controlled by Arduino and Odroid [24]. The Arduino is used for achieving the velocity control and receiving low-level sensor information like battery power remain. Odroid is a powerful microprocessor installed with Ubuntu OS. Arduino will keep uploading all the information to the Odroid, at the same time, it will listen to any velocity command given by the Odroid. Currently, the vehicle is implemented with the PID velocity control which is using linear (V) and angular (ω) velocity as the inputs to command the vehicle. In

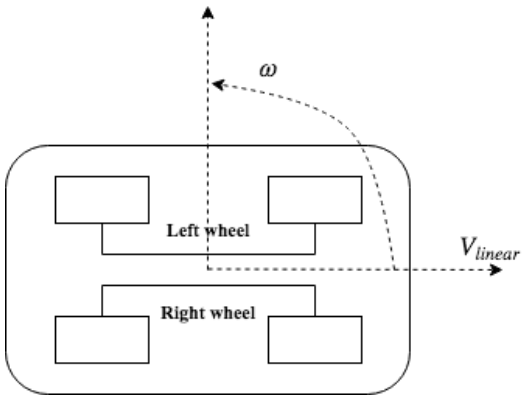

Figure 14: Ground vehicle diagram

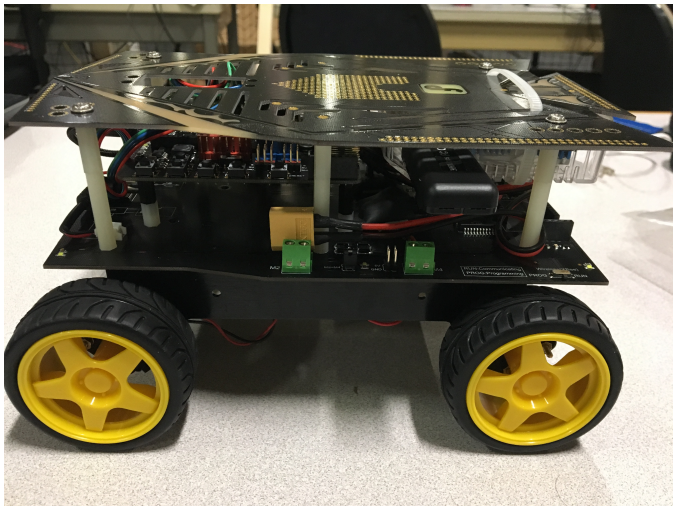

Figure 15: The self-developed UGV

Direction	linear Velocity	Angular Velocity
Forward	0.1m/s	0
Backward	-0.1m/s	0
Left	0	0.5 rad/s
right	0	-0.5 rad/s

Table 2: Velocity setting

the Odroid, Robot Operating System (ROS) software is used for managing all the projects/packages and communication. The user uses the C++ or Python programming language to create their own project based on pre-implemented libraries and tools. Different projects can communicate with each other through ROS topics. For this research, Arduino communication is an independent package which is for transferring data between Arduino and ROS. This package will create a velocity command listener under the ROS-topic, and any incoming command published on this topic will be transmitted to Arduino. Then, another package is created for receiving the recognition result from GUI, computing the desired velocity, and publishing it to the ROS-topic. Here, TCP/IP technology is used to transmitted the recognition result wirelessly from another computer (the one connect with Kinect) to the Odroid on the vehicle. These two computers are required to work under the same Wi-Fi network. Once the Odroid received the signal, the Velocity will be set based on the directions as specified in table 2.

On the Kinect side, whenever the Python GUI finishes one real-time recognition, it will send the direction result to the Odroid by TCP/IP. Since each gesture pattern needs to contain 3 seconds, adding with gesture changing time and recognition time, totally it will take about 4 seconds to give one Velocity command. In order to prevent the vehicle goes too far in this 4 seconds, a stop command will be sent after 2 seconds of the directional command.

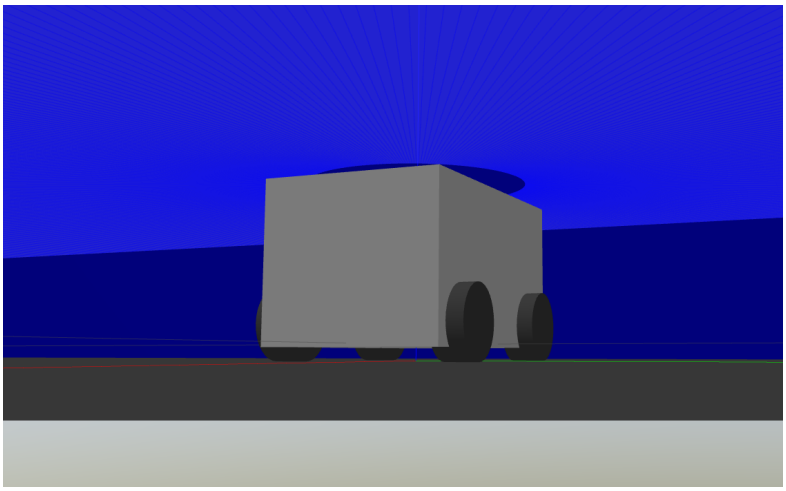

Figure 16: Gazebo simulated UGV

4.3.2 Gazebo Simulation

Gazebo is a well-designed simulator mainly for robot simulation. The goal for using the simulator is to build a four-wheel drive which try to imitate the real ground vehicle and test the deterministic learning algorithm in a more convenient way. The simulated UGV is simply built up by one rectangular box (body) with four cylinders (wheels), each cylinder is connected to the body with one revolute joint (Figure 16). Then this simulated car can move freely by adding forces on these four joints. Gazebo has a plugin called "differential drive controller" which is especially aimed for controlling this type of ground vehicles. This plugin will take the vehicle's parameters like wheel radius and vehicle weight, to compute the desired force on the wheels from the linear and angular velocity commands. Since the Gazebo is running under the ROS, the plugin will create a topic under ROS-topic list which allows users to command the vehicle.

For the simulation testing,

4.4 Real-time Controlling Result

We asked for different persons to do the gestures in front of the Kinect to test the real-time control. First, I demonstrated all four gestures to each person once.

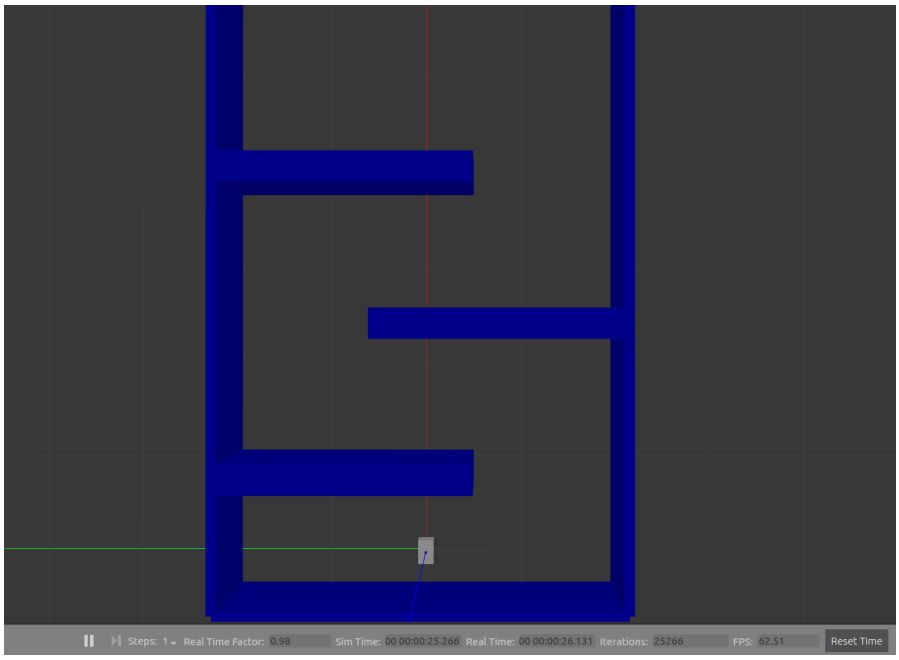

Figure 17: Gazebo simulated UGV with simple maze

Then, they try to imitate the gestures (keep waving their hand) in front of the Kinect. The recognition result and the ground vehicle gave the desired result. We have included some experimental results from Figure 18 to 21 for better illustration. The first column graphs are showing the gesture motion. The second column is showing the intention-decoded results. The last column is showing the actual ground vehicle moving routines, from the first graph of moving forward (Figure 18), then turning left (Figure 19), turning right (Figure 20), and finally moving backward (Figure 21).

To test the performance of the gesture recognition on the UGV Control. A simple maze was built for both Gazebo (Figure 17) and the real world (Figure 22). The goal of the maze is to test the efficiency of this gesture control. In order to prevent the car from hitting the wall, a laser scanner is used to detect obstacle around the UGV. Then, a simply shared control algorithm is implemented into the UGV. If the UGV is too close ($<$ 25 cm) to the obstacle in the front, the linear

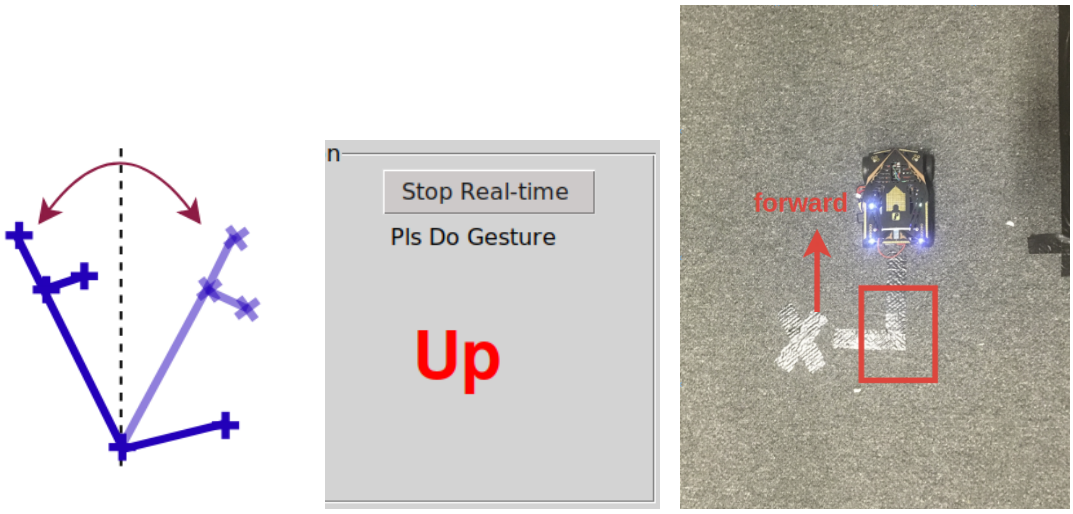

Figure 18: When presenting forward gesture during the real-time control

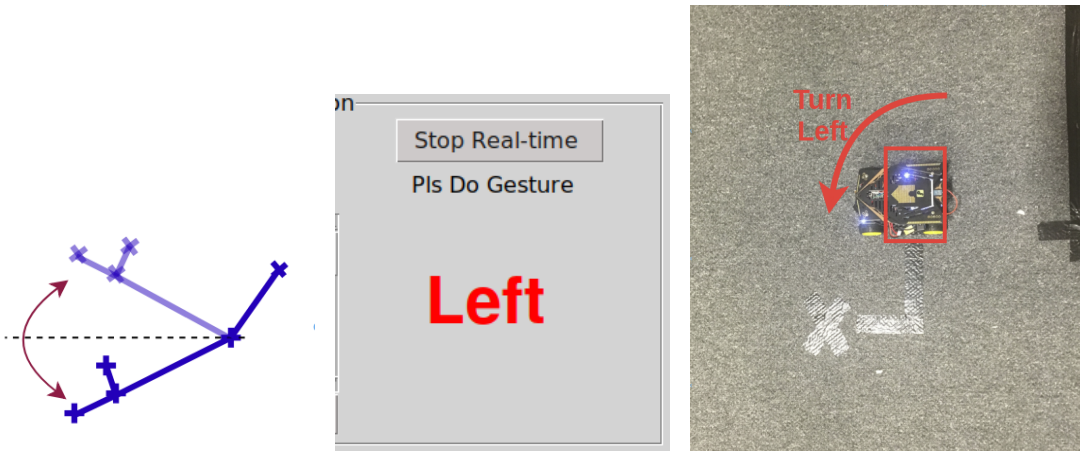

Figure 19: When presenting left gesture during the real-time control

Figure 20: When presenting right gesture during the real-time control

Figure 21: When presenting backward gesture during the real-time control

velocity will be held at zero, any forward command is received from the gesture recognition will be ignored (same idea for moving backward). Other than that, the gesture command will have the full control of the UGV.

One of the example diagrams of the UGV moving is performed as Figure 24. The red array is representing the UGV with the array pointing direction as the forward direction. Although in Section 4.2, the recognition rate is examined to be 100%, during the real-time control, the recognized direction is not always the same as the desired one. When the person is standing in front of the camera and looking at the UGV, it takes time for the person to think about which direction

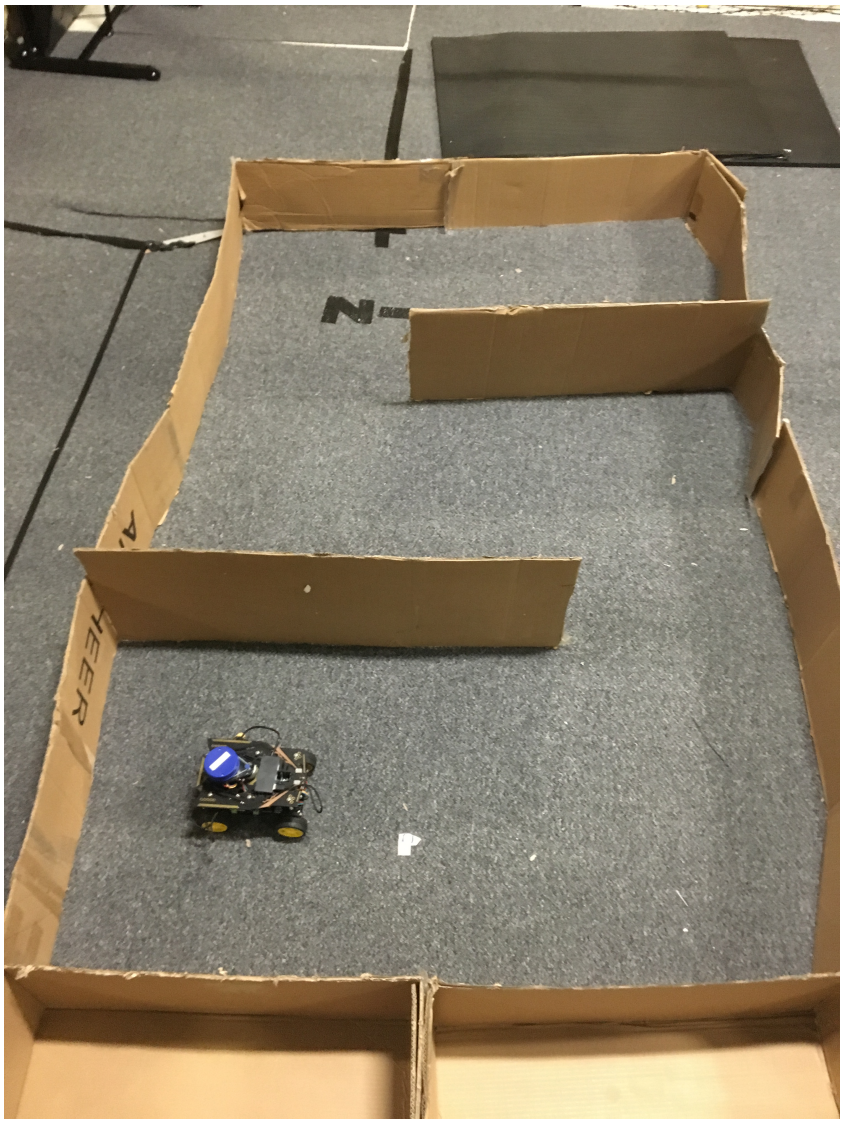

Figure 22: A simply real-world maze for gesture control efficiency test

to command and changing one gesture to another. The recognition period which including those invalid signals will have much lower recognition rate than desired. That's the reason causing the UGV to become too close to the wall.

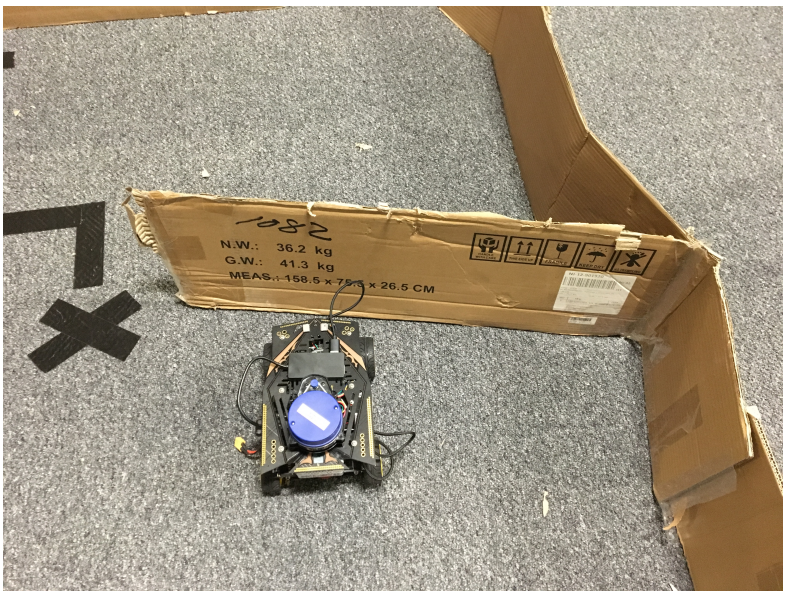

Figure 23: The Lidar sensor will prevent the UGV from hitting to the wall

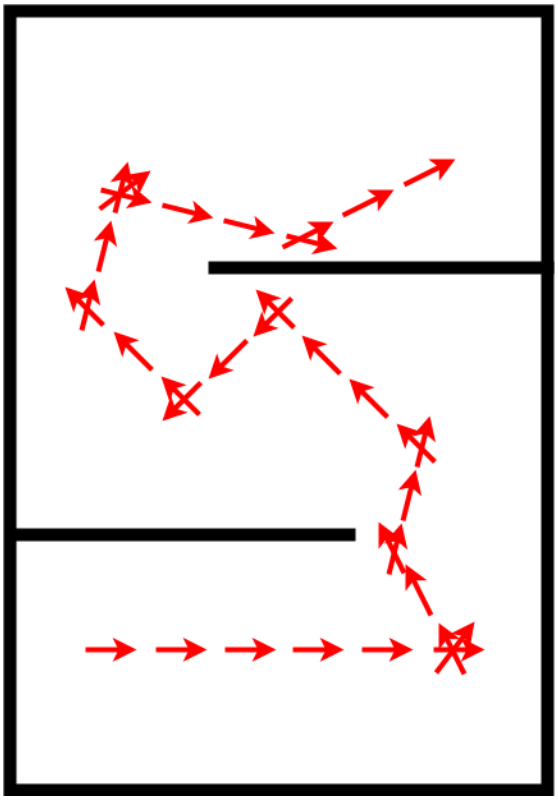

Figure 24: The diagram for one example of the UGV trajectory under the shared control algorithm. The red array is representing the UGV with the array pointing direction as the forward direction.

CHAPTER 5

Conclusion

A novel method of robotic direction control through human gesture recognition is presented in this thesis. By using the Kinect for gathering human skeleton tracking data, we define four features including joint position, angle, linear and angular velocities to train the RBF NNs by utilizing the discrete-time deterministic learning algorithm. Based on the training results, real-time recognition of the arm gestures and human intention decoding can be realized, and such intention can be translated to four directional commands (moving left, right, forward, and backward), which are sent to command the ground vehicle. All the process is designed into the graphical user interface for the user to conveniently design their own gestures and controlling the robot. In the experiment, by testing the real-time controlling with random people, the ground vehicle can correctly follow the instructions (human intention), which strongly demonstrate the effectiveness of our proposed approach.

LIST OF REFERENCES

[1] P. Wang, Z. Li, Y. Hou, and W. Li, "Action recognition based on joint trajectory maps using convolutional neural networks," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 102–106.

[2] T. B. Sheridan, "Human–robot interaction: status and challenges," *Human factors*, vol. 58, no. 4, pp. 525–532, 2016.

[3] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[4] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015.

[5] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," in *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1565–1569.

[6] D. Xu, X. Wu, Y.-L. Chen, and Y. Xu, "Online dynamic gesture recognition for human robot interaction," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 3-4, pp. 583–596, 2015.

[7] F. Liu, W. Zeng, C. Yuan, Q. Wang, Y. Wang, and B. Lu, "Trajectory-based hand gesture recognition using kinect via deterministic learning," in *2018 37th Chinese Control Conference (CCC)*. IEEE, 2018, pp. 9273–9278.

[8] J. Lu, G. Wang, and P. Moulin, "Human identity and gender recognition from gait sequences with arbitrary walking directions," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 51–61, 2014.

[9] M. Deng, C. Wang, and Q. Chen, "Human gait recognition based on deterministic learning through multiple views fusion," *Pattern Recognition Letters*, vol. 78, pp. 56–63, 2016.

[10] W. Zeng and C. Wang, "Human gait recognition via deterministic learning," *neural networks*, vol. 35, pp. 92–102, 2012.

[11] R. Xie and J. Cao, "Accelerometer-based hand gesture recognition by neural network and similarity matching," *IEEE Sensors Journal*, vol. 16, no. 11, pp. 4537–4545, 2016.

[12] H. Cheng, L. Yang, and Z. Liu, "Survey on 3d hand gesture recognition." *IEEE Trans. Circuits Syst. Video Techn.*, vol. 26, no. 9, pp. 1659–1673, 2016.

[13] R. Lun and W. Zhao, "A survey of applications and human motion recognition with microsoft kinect," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 05, p. 1555008, 2015.

[14] R. Ibanez, A. Soria, A. Teyseyre, and M. Campo, "Easy gesture recognition for kinect," *Advances in Engineering Software*, vol. 76, pp. 171–180, 2014.

[15] Y. Chen, Z. Ding, Y.-L. Chen, and X. Wu, "Rapid recognition of dynamic hand gestures using leap motion," in *Information and Automation, 2015 IEEE International Conference on*. IEEE, 2015, pp. 1419–1424.

[16] C. Yuan and C. Wang, "Design and performance analysis of deterministic learning of sampled-data nonlinear systems," *Science China Information Sciences*, vol. 57, no. 3, pp. 1–18, 2014.

[17] C. Yuan, "Persistency of excitation and performance analysis of deterministic learning algorithms," *Master Degree Thesis of South China University of Technology*, 2012.

[18] C. Wang and D. J. Hill, "Learning from neural control," *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 130–146, 2006.

[19] C. Wang and D. J. Hill, "Deterministic learning and rapid dynamical pattern recognition," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 617–630, 2007.

[20] J. Park and I. W. Sandberg, "Approximation and radial-basis-function networks," *Neural computation*, vol. 5, no. 2, pp. 305–316, 1993.

[21] D. Gorinevsky, "On the persistency of excitation in radial basis function network identification of nonlinear systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1237–1244, 1995.

[22] C. Wang and D. J. Hill, *Deterministic learning theory for identification, recognition, and control*. CRC Press, 2009.

[23] L. P. Shil'nikov, *Methods of qualitative theory in nonlinear dynamics*. World Scientific, 2001, vol. 5.

[24] L. Hardkernel Co. "Odroid." 2015. [Online]. Available: https://www.hardkernel.com/

APPENDIX

Python coding

A.1 Deterministic Learning

```
#!/user/bin/env python
import matplotlib.pyplot as plt
import numpy as np
import RBFNN


def Det_learn(file_addr,filename):
    ''' feature creating four
    '''
    feature = np.load(file_addr)
    if feature.shape[1] > 10:
        feature = feature.T
    '''cent nerual network creating
        all four from -1.2 to 1.2
    '''
    cent = np.load('cent.npy')
    cent_length = cent.shape[1]
    '''feature learning repeat 150 times
    '''
    x1=[]
    x2=[]
    x3=[]
    x4=[]
    kk = 300
```

```
'''stack feature 300 times
'''
x1 = np.tile(feature[:,0],kk)
x2 = np.tile(feature[:,1],kk)
x3 = np.tile(feature[:,2],kk)
x4 = np.tile(feature[:,3],kk)
st_steps = x1.shape[0]
x_repeat = np.array([x1,x2,x3,x4])

#parameters setup
Ts = 1./30
eta=0.15
a=0.5
alpha=1.5#1.5
sigma=10 #10

pj = 1000  # steps to plot converge
s1 = np.zeros(cent_length)
s2 = np.zeros(cent_length)



s1 = RBFNN.S(x_repeat[:,0],eta)

s1_update = s1/(1+sigma* np.sum(s1**2))
W= np.zeros([4,cent_length])
x_hat = np.zeros([4,st_steps])
```

```
    V=[]
    id = 0
    for i in range(1,st_steps-1):
        s2 = RBFNN.S(x_repeat[:,i],eta)

        error_cur = x_hat[:,i]-x_repeat[:,i]
        error_pre = x_hat[:,i-1]-x_repeat[:,i-1]

        U = W - np.reshape(alpha*sigma*(error_cur-a*(error_pre))...
            ,(4,1))*s1_update #4by8
        x_hat[:,i+1]=x_repeat[:,i]+a*error_cur+Ts*np.dot(U,s2)
        W = U
        if i > st_steps-pj-2:
            V.append(U)
        s1=s2
        s1_update = s1/(1+sigma* np.sum(s1**2))
    W_bar = np.mean(V,axis = 0)
    np.save('W/'+filename+'_W.npy',W_bar)
    np.save('temp/xhat.npy',x_hat)
'''save the converging result
    V1= np.asarray(V)
    np.save('temp/V3.npy',V1)
'''
```

A.2 Recognition

```
#!/user/bin/env python
import matplotlib.pyplot as plt
```

```
import numpy as np
import os
import RBFNN
import timeit

def Det_recog(pattern_name,w_list):
    feature = np.load(pattern_name)
    w_n = len(w_list)   # number of Weight file to compare
    k = feature.shape[0]  # feature length
    print(k)
    if k >100:    # reduce the feature length at the beginning
        feature = feature[k-100:k,:]
        k=feature.shape[0]
    U = np.zeros([4*w_n,k])
    ### parameters setup
    Ts = 1./300
    b = -150
    eta = 0.15
    Te= 10 # beginning skip length (should smaller than k)
    error=np.zeros([w_n,4])

    w = np.empty((4*w_n,83521))
    for i in range(w_n):
        w[i*4:i*4+4,:] = np.load("W/"+w_list[i])

    for i in range(k-1):
```

```
    U[:,i+1]=(1+b*Ts)*U[:,i]+np.dot(w,RBFNN.S(feature[i,:]...
          ,eta))-np.tile((feature[i+1,:]-feature[i,:])/Ts,w_n)

error= np.sum(abs(U[:,Te:k]),axis=1)/Te
error = np.reshape(error,(w_n,4))
error = np.sum(error, axis = 1)

U = np.sum(abs(U.reshape(w_n,4,k)),axis=1)
np.save("temp/error.npy",U)
result = []
for j in range(3):
    index = np.argmin(error)
    result.append(w_list[index])
    #print(error[index])
    error[index] = error[index]+2999

with open("temp/result.txt", 'w') as f:
    for s in result:
        f.write(s + '\n')

return(result)
```

BIBLIOGRAPHY

Chen, Y., Ding, Z., Chen, Y.-L., and Wu, X., "Rapid recognition of dynamic hand gestures using leap motion," in *Information and Automation, 2015 IEEE International Conference on.* IEEE, 2015, pp. 1419–1424.

Cheng, H., Yang, L., and Liu, Z., "Survey on 3d hand gesture recognition." *IEEE Trans. Circuits Syst. Video Techn.*, vol. 26, no. 9, pp. 1659–1673, 2016.

Deng, M., Wang, C., and Chen, Q., "Human gait recognition based on deterministic learning through multiple views fusion," *Pattern Recognition Letters*, vol. 78, pp. 56–63, 2016.

Gorinevsky, D., "On the persistency of excitation in radial basis function network identification of nonlinear systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1237–1244, 1995.

Hardkernel Co., L. "Odroid." 2015. [Online]. Available: https://www.hardkernel.com/

Ibanez, R., Soria, A., Teyseyre, A., and Campo, M., "Easy gesture recognition for kinect," *Advances in Engineering Software*, vol. 76, pp. 171–180, 2014.

Liu, F., Zeng, W., Yuan, C., Wang, Q., Wang, Y., and Lu, B., "Trajectory-based hand gesture recognition using kinect via deterministic learning," in *2018 37th Chinese Control Conference (CCC).* IEEE, 2018, pp. 9273–9278.

Lu, J., Wang, G., and Moulin, P., "Human identity and gender recognition from gait sequences with arbitrary walking directions," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 51–61, 2014.

Lun, R. and Zhao, W., "A survey of applications and human motion recognition with microsoft kinect," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 05, p. 1555008, 2015.

Marin, G., Dominio, F., and Zanuttigh, P., "Hand gesture recognition with leap motion and kinect devices," in *Image Processing (ICIP), 2014 IEEE International Conference on.* IEEE, 2014, pp. 1565–1569.

Park, J. and Sandberg, I. W., "Approximation and radial-basis-function networks," *Neural computation*, vol. 5, no. 2, pp. 305–316, 1993.

Rautaray, S. S. and Agrawal, A., "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015.

Sheridan, T. B., "Human–robot interaction: status and challenges," *Human factors*, vol. 58, no. 4, pp. 525–532, 2016.

Shil'nikov, L. P., *Methods of qualitative theory in nonlinear dynamics*. World Scientific, 2001, vol. 5.

Siciliano, B. and Khatib, O., *Springer handbook of robotics*. Springer, 2016.

Toolan, T. M. and Tufts, D. W., "Detection and estimation in non-stationary environments," in *Proceedings IEEE Asilomar Conference on Signals, Systems & Computers*, Nov. 2003, pp. 797–801.

Wang, C. and Hill, D. J., "Learning from neural control," *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 130–146, 2006.

Wang, C. and Hill, D. J., "Deterministic learning and rapid dynamical pattern recognition," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 617–630, 2007.

Wang, C. and Hill, D. J., *Deterministic learning theory for identification, recognition, and control*. CRC Press, 2009.

Wang, P., Li, Z., Hou, Y., and Li, W., "Action recognition based on joint trajectory maps using convolutional neural networks," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 102–106.

Xie, R. and Cao, J., "Accelerometer-based hand gesture recognition by neural network and similarity matching," *IEEE Sensors Journal*, vol. 16, no. 11, pp. 4537–4545, 2016.

Xu, D., Wu, X., Chen, Y.-L., and Xu, Y., "Online dynamic gesture recognition for human robot interaction," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 3-4, pp. 583–596, 2015.

Yuan, C., "Persistency of excitation and performance analysis of deterministic learning algorithms," *Master Degree Thesis of South China University of Technology*, 2012.

Yuan, C. and Wang, C., "Design and performance analysis of deterministic learning of sampled-data nonlinear systems," *Science China Information Sciences*, vol. 57, no. 3, pp. 1–18, 2014.

Zeng, W. and Wang, C., "Human gait recognition via deterministic learning," *neural networks*, vol. 35, pp. 92–102, 2012.